Weighted Model Counting Without Parameter Variables

16th February 2021

1 Introduction

Notation. For any propositional formula ϕ and $p, q \in \mathbb{R}$, let $[\phi]^p_q \colon 2^X \to \mathbb{R}$ be a pseudo-Boolean function defined as

$$[\phi]^p_q(Y) \coloneqq \begin{cases} p & \text{if } Y \models \phi \\ q & \text{otherwise} \end{cases}$$

for any $Y \subseteq X$.

Lemma 1. *An ADD with n variables has $\mathcal{O}(2^n)$ nodes.*

Proof. An ADD with n variables can be, at most, a complete binary tree of height $n+1$ (as measured by the number of vertices). It would then have $2^0 + 2^1 + \cdots + 2^n = 2^{n+1} = \mathcal{O}(2^n)$ nodes. ☐

Lemma 2. *Let ϕ be a conjunction of n literals. Then the ADD representation of $[\phi]^p_q$ can be constructed in $\mathcal{O}(2^n)$ time for any $p, q \in \mathbb{R}$ such that $p \neq q$.*

Proof. The ADD representation of ϕ itself can be constructed with a sequence of $n-1$ calls to `apply` with one of the two operands in each call always a literal. The number of variables in the other operand then follows the sequence $1, 2, 3, \dots, n-1$. By Lemma 1, the numbers of nodes in the ADD representations of these operands is then $\mathcal{O}(2^1), \mathcal{O}(2^2), \dots, \mathcal{O}(2^{n-1})$. Since one of the operands is of constant size, the overall time complexity of all calls to `apply` is then

$$\mathcal{O}(2^1) + \mathcal{O}(2^2) + \cdots + \mathcal{O}(2^{n-1}) = \mathcal{O}(2^n).$$

Let α be the ADD representation of ϕ. Then the ADD representation of $[\phi]^p_q$ is $(p-q)\alpha + q$. As scalar operations can obviously be implemented in linear time, the overall complexity remains $\mathcal{O}(2^n)$. ☐

Definition 1 (old WMC instance)**.** An old WMC instance is a tuple (ϕ, X_I, X_P, w), where X_I is the set of indicator variables, X_P is the set of parameter variables, ϕ is a propositional formula over $X_I \cup X_P$, and $w \colon X_P \to \mathbb{R}$ is the weight function.

Definition 2 (new WMC instance)**.** A new WMC instance is a tuple (F, X, ω), where X is the set of variables, F is a set of pseudo-Boolean functions $2^X \to \mathbb{R}$, and $\omega \in \mathbb{R}$ is the scaling factor.

References.

- Related work without publicly available implementations:
  - direct compilation to SDDs [2]
  - direct compilation to PSDDs, also eliminating parameter variables (a thesis)
  - maybe two more papers

Notes.

- Apparently, the DPMC paper already shows that taking the first offered decomposition tree is best.
- It is already well-known that WMC is FPT.

2 Parameter Variable Elimination

Notes.

- Let X_P be the set of parameter variable and X_I be the set of indicator variables.
- Parameter variables are either taken from the LMAP file (for encodings produced by Ace) or assumed to be the variables that have both weights equal to 1.
- If a parameter variable in a clause is 'negated', we can ignore the clause. We assume that there are no clauses with more than one instance of parameter variables.
- The second **foreach** loop can be performed in constant time by representing ϕ' as a list and assuming that the two 'clauses' are adjacent in that list (and incorporating it into the first loop).
- The d map is constructed in $\mathcal{O}(|X_P| \log |X_P|)$ time (we want to use a data structure based on binary search trees rather than hashing).
- `rename` can be implemented in $\mathcal{O}(\log |X_P|)$ time.
- This may look like preprocessing, but all the transformations are local and thus can be incorporated into an encoding algorithm with no slowdown. In fact, if anything, the resulting algorithm would be slightly faster, as it would have less data to output.

3 Parameterised Complexity of DPMC

Notes.

- Summary of results
  - We establish DPMC inference as fixed-parameter tractable.
  - We experimentally show that DPMC is best on low-to-moderate treewidth instances, and cd06+c2d overtakes on higher treewidth instances.
- By DPMC, we always mean DMC+lg.

TODO

- do a literature search focused around these papers
- do I need to formally consider extending a pseudo-Boolean function to a bigger domain?
- Introduce all the notation surrounding graphs ($\mathcal{V}$, $\mathcal{L}$, $\mathcal{C}$, etc.) (and be consistent with its usage)
- Define:
  - scalar operations on ADDs
  - an ADD as a DAG.

Algorithm 1: WMC instance transformation

```
Data: an (old-format) WMC instance $(\phi, X_I, X_P, w)$
Result: a (new-format) WMC instance $(F, X, \omega)$
$F \leftarrow \emptyset$;
$\omega \leftarrow 1$;
let $d\colon X_P \to \mathbb{N}$ be defined as $v \mapsto |\{u \in X_P \mid u \leq v\}|$;
foreach clause $c \in \phi$ do
    if $c \cap X_P = \{v\}$ for some $v$ and $w(v) \neq 1$ then
        if $|c| = 1$ then
            $\omega \leftarrow \omega \times w(v)$;
        else
            $F \leftarrow F \cup \left\{ \left[ \bigwedge_{l \in c \setminus \{v\}} \neg l \right]_1^{w(v)} \right\}$;
    else if $\{v \mid \neg v \in c\} \cap X_P = \emptyset$ then
        $F \leftarrow F \cup \{[c]_0^1\}$;
foreach indicator variable $v \in X_I$ do
    if $\{[v]_1^p, [\neg v]_1^q\} \subseteq F$ for some $p$ and $q$ then
        $F \leftarrow F \setminus \{[v]_1^p, [\neg v]_1^q\} \cup \{[v]_q^p\}$;
replace every variable $v$ in $F$ with rename($v$);
return $(F, X_I, \omega)$;
Function rename($v$):
    $S \leftarrow \{u \in X_P \mid u \leq v\}$;
    if $S = \emptyset$ then return $v$;
    return $v - d(\max S)$;
```

- What does it mean for an ADD to 'have' variables? Maybe refer to pseudo-Boolean function sensitivity.
- Formal definition of a previous WMC instance (CNF, literal weight function) and the new definition (set of $2^X \to \mathbb{R}_{\ge 0}$ pseudo-Boolean functions and a constant). Note that the constant idea is borrowed from `bklm16`.
- Boolean formula in CNF (perhaps this is too trivial to define),
- primal graph of a CNF formula (a.k.a. Gaifman/(variable) interaction/connectivity/clique/representing graph),
- Bayesian network (I already have a definition of these last two),
- moralisation of a Bayesian network (or of any DAG),

Theorem 1 ([4], rephrased)**.** *BN Inference (for all algorithms that accept arbitrary instances) has a lower bound that's linear in the size of the BN and exponential in the treewidth of its moralisation (provide the exact formula).*

Definition 3 ([3], with some changes)**.** Let X be a set of Boolean variables, and F be a set of pseudo-Boolean functions $2^X \to \mathbb{R}$. A *project-join tree* (PJT) of F is a tuple (T, r, γ, π) where:

- T is a tree with root $r \in \mathcal{V}(T)$,
- $\gamma\colon \mathcal{L}(T) \to F$ is a bijection between the leaves of T and the pseudo-Boolean functions in F, and
- $\pi\colon \mathcal{V}(T) \setminus \mathcal{L}(T) \to 2^X$ is a labelling function on internal nodes.

Moreover, (T, r, γ, π) must satisfy the following two properties:

1. $\{\pi(n) : n \in \mathcal{V}(T) \setminus \mathcal{L}(T)\}$ is a partition of X, and
2. for each internal node $n \in \mathcal{V}(T) \setminus \mathcal{L}(T)$, variable $x \in \pi(n)$, and 'clause' $c \in \operatorname{im}\gamma$ such that x appears in c, the leaf node $\gamma^{-1}(c)$ must be a descendant of n in T.

Definition 4 ([3])**.**

$$\texttt{Vars}(n) \coloneqq \begin{cases} \texttt{Vars}(\gamma(n)) & \text{if } n \in \mathcal{L}(T) \\ \left(\bigcup_{o \in \mathcal{C}(n)} \texttt{Vars}(o)\right) \setminus \pi(n) & \text{if } n \notin \mathcal{L}(T). \end{cases}$$

$$\texttt{size}(n) \coloneqq \begin{cases} |\texttt{Vars}(n)| & \text{if } n \in \mathcal{L}(T) \\ |\texttt{Vars}(n) \cup \pi(n)| & \text{if } n \notin \mathcal{L}(T). \end{cases}$$

(Note that $\texttt{size}(n)$ is the number of variables that can appear during the computation of $\delta(n)$ (excluding recursive calls).) The *width* of a PJT (T, r, γ, π) is $\texttt{width}(T) \coloneqq \max_{n \in \mathcal{V}(T)} \texttt{size}(n)$.

Theorem 2 ([3], rephrased, 'ADD width is equal to the treewidth of the primal graph')**.** *Given a CNF formula ϕ with a tree decomposition of its primal graph of width w, Algorithm 2 (from [3]) returns a PJT of ϕ of width at most $w + 1$.*

Definition 5 ([5], rephrased)**.** A *tree decomposition* of a graph G is a pair (T, χ), where T is a tree and $\chi\colon \mathcal{V}(T) \to 2^{\mathcal{V}(G)}$ is a labelling function, with the following properties:

- $\bigcup_{t \in \mathcal{V}(T)} \chi(t) = \mathcal{V}(G)$;
- for every edge $e \in \mathcal{E}(G)$, there exists $t \in \mathcal{V}(T)$ such that e has both endpoints in $\chi(t)$;
- for all $t, t', t'' \in \mathcal{V}(T)$, if t' is on the path between t and t'', then $\chi(t) \cap \chi(t'') \subseteq \chi(t')$.

The *width* of tree decomposition (T, χ) is $\max_{t \in \mathcal{V}(T)} |\chi(t)| - 1)$. The *treewidth* of graph G is the smallest w such that G has a tree decomposition of width w.

Definition 6. Let $f, g\colon 2^X \to \mathbb{R}$ be pseudo-Boolean functions. Operations such as addition and multiplication are defined pointwise as

$$(f+g)(Y) \coloneqq f(Y) + g(Y),$$

and

$$(fg)(Y) \coloneqq f(Y)g(Y)$$

for all $Y \subseteq X$.

Definition 7. Let $f\colon 2^X \to \mathbb{R}$ be a pseudo-Boolean function, and $x \in X$. Then $f|_{x=0}, f|_{x=1}\colon 2^X \to \mathbb{R}$ are *restrictions* of f defined as

$$f|_{x=0}(Y) \coloneqq f(Y \setminus \{x\}),$$

and

$$f|_{x=1}(Y) \coloneqq f(Y \cup \{x\})$$

for all $Y \subseteq X$.

Definition 8. Let X be a set. For any $x \in X$, *projection* $\exists_x$ is an endomorphism $\exists_x\colon \mathbb{R}^{2^X} \to \mathbb{R}^{2^X}$ defined as

$$\exists_x f = f|_{x=1} + f|_{x=0}$$

for any $f\colon 2^X \to \mathbb{R}$.

Lemma 3. *Let $f, g\colon 2^X \to \mathbb{R}$ be pseudo-Boolean functions represented by ADDs with n and m nodes, respectively, and $x \in X$. Then $f+g$ and fg can be computed in $\mathcal{O}(mn)$ time, and $\exists_x f$ can be computed in $\mathcal{O}(n^2)$ time.*

Proof. Addition and multiplication are implemented by `apply` algorithm which takes $\mathcal{O}(mn)$ time [1]. Projection consists of two restrictions and an addition by Definition 8. The computational complexity of restriction is dominated by the reduction operation that transforms a decision diagram into a minimal canonical form [1]. While the original reduction algorithm had a $\mathcal{O}(n \log n)$ complexity, caching can reduce it to $\mathcal{O}(n)$ [6]. Either way, the complexity of projection is still $\mathcal{O}(n^2)$. □

Definition 9. Let (T, r, γ, π) be a PJT, and X be the set of variables. The functionality of DPMC execution can be represented by $\delta(r)$, where $\delta\colon \mathcal{V}(T) \to \mathbb{R}^{2^X}$ is a recursive function defined as

$$\delta(t) = \begin{cases} \gamma(t) & \text{if } t \in \mathcal{L}(T) \\ \exists_{\pi(t)} \prod_{u \in \mathcal{C}(t)} \delta(u) & \text{otherwise.} \end{cases} \tag{1}$$

The range of $\delta(r)$ then contains a single real number, i.e., the answer.

Theorem 3 (Theorem 4 in [3], almost verbatim)**.** *Let ϕ be a CNF formula over a set X of variables and (S, χ) be a tree decomposition of the primal graph of ϕ of width w. Then Algorithm 2 returns a PJT of ϕ of width at most $w+1$.*

Theorem 4. *Let (T, r, γ, π) be a PJT of width k. Then* DPMC *execution is fixed-parameter tractable with respect to the k. Specifically,* DPMC *execution time complexity is $\mathcal{O}(4^k nm)$, where $n = |\mathcal{L}(T)|$ is the number of clauses/leaves, and m is the number of variables.*

Proof.
- Let us consider the overall complexity of all multiplications and projections throughout the recursive calls of δ in Definition 9. Clearly, the overall complexity is the sum of the complexity of operations performed within each call to δ.
- Let $t \in \mathcal{L}(T)$ be a leaf. Note that $|\texttt{Vars}(\gamma(t))| \le k$ by Definition 4. Assuming that clauses have no repeating variables, it takes $\mathcal{O}(2^k)$ time to construct $\delta(t)$ by Lemma 2 and $\mathcal{O}(2^k n)$ time to construct all leaves of the PJT.

- Now let $t \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ be an internal vertex, in which case we perform $|\mathcal{C}(t)| - 1$ multiplications and $|\pi(t)|$ projections. Note that $|\mathcal{C}(t)| \leq |\mathcal{L}(T)| = n$, $|\pi(t)| \leq k$, and the ADDs involved can have up to k variables. This means that each ADD has $\mathcal{O}(2^k)$ nodes (regardless of whether it comes from $\mathcal{C}(t)$ or from multiplications). Each such multiplication then takes $\mathcal{O}(4^k)$ time by Lemma 3, and so all multiplications will take $\mathcal{O}(4^k n)$ time for t and $\mathcal{O}(4^k nm)$ time across all $t \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ since the number of vertices in the PJT is bounded by the number of variables.

- Each variable is projected exactly once and is always projected from an ADD with at most $\mathcal{O}(2^k)$ nodes. Thus, the time complexity of projecting all variables is $\mathcal{O}(4^k m)$.

- In total, we get $\mathcal{O}(2^k n)$ time for the leaves, $\mathcal{O}(4^k nm)$ for multiplications, and $\mathcal{O}(4^k m)$ for projections, resulting in $\mathcal{O}(4^k nm)$ time in total.

□

References

[1] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers 35*, 8 (1986), 677–691.

[2] Choi, A., Kisa, D., and Darwiche, A. Compiling probabilistic graphical models using sentential decision diagrams. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, July 8-10, 2013. Proceedings* (2013), L. C. van der Gaag, Ed., vol. 7958 of *Lecture Notes in Computer Science*, Springer, pp. 121–132.

[3] Dudek, J. M., Phan, V. H. N., and Vardi, M. Y. DPMC: weighted model counting by dynamic programming on project-join trees. In *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings* (2020), H. Simonis, Ed., vol. 12333 of *Lecture Notes in Computer Science*, Springer, pp. 211–230.

[4] Kwisthout, J., Bodlaender, H. L., and van der Gaag, L. C. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings* (2010), H. Coelho, R. Studer, and M. J. Wooldridge, Eds., vol. 215 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 237–242.

[5] Robertson, N., and Seymour, P. D. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B 36*, 1 (1984), 49–64.

[6] Somenzi, F. CUDD: CU decision diagram package release 3.0.0. *University of Colorado at Boulder* (2015).